



From Haskell via C_ω to LINQ

A Personal Perspective
Erik Meijer

Presentation

Data

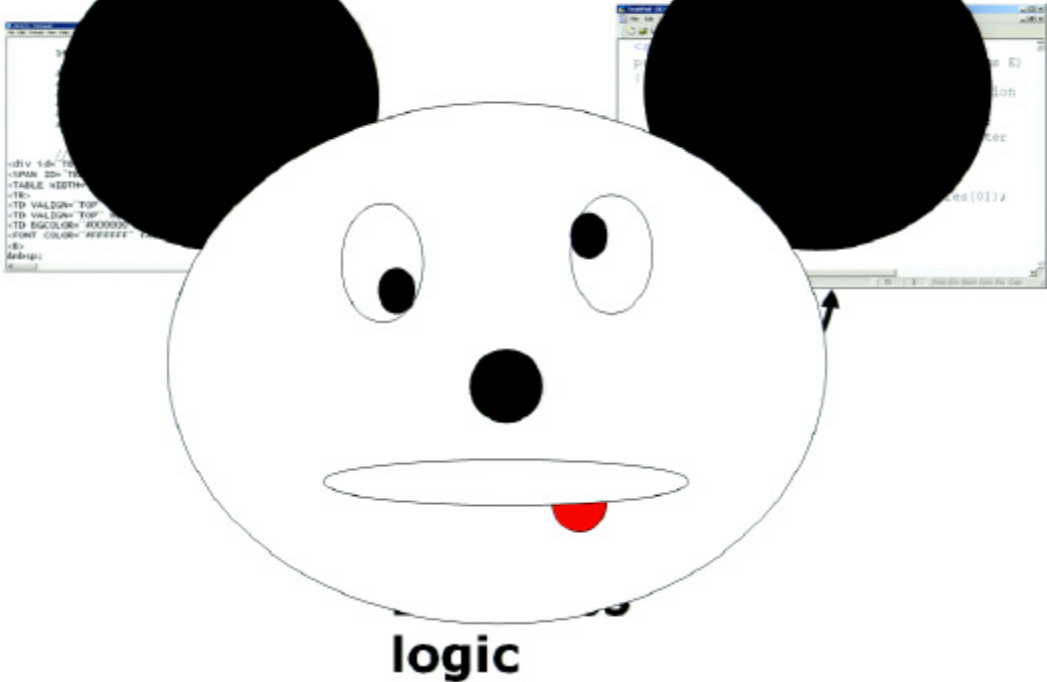
```
<!--[[[
setICHeaderC:\library\shared\toolbar\graphics\bin
setICHeaderC:\aspgo\bin
addICMenu("Menu") "MSDN Home" "default.asp"
addICMenu("Menu") "MSDN Library" "library/"
addICMenu("Menu") "Downloads" "downloads/"
addICMenu("Menu") "Code Center" "code/defa
addICMenu("Menu") "Site Map" "sitemap/defa
addICMenu("Menu") "MSDN worldwide" "worldw
]]]-->
</SCRIPT> </HEAD> <BODY TOPMARGIN="0" LEFTM
<div id="tbcContainer" style="height:80px;">
<SPAN id="tbcDownLevelDiv">
<TABLE WIDTH="100%" CELLSPACING="0" CELLSPACING="0" BORDER
<tr>
<td VALIGN="TOP" HEIGHT="90" ROWSPAN="2"><A HREF="/default
<td VALIGN="TOP" HEIGHT="20" ALIGN="RIGHT">
</font>
</td>
</tr>
</table>
</div>
```

```
<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E)
{
    SqlConnection myConnection = new SqlConnection
    ("server=localhost;" +
     "database=pubs;Trusted_Connection=yes;");
    SqlDataAdapter myCommand = new SqlDataAdapter
    ("SELECT * FROM Authors", myConnection);
    DataSet ds = new DataSet();
    myCommand.Fill(ds);
    DataView source = new DataView(ds.Tables(0));
    MyDataGrid.DataSource = source;
    MyDataGrid.DataBind();
}
</script>
```

```
<% Page Language="Vb" ASPCompat="true" %>
<script runat="server">
Dim myComp
Public Sub Page_Load()
    myComp = New MyBTAComponent()
    myComp.Name = "Bob"
End Sub
</script>
<html>
<body>
    Response.Write myComp.SayHello()
</body>
</html>
```

Business logic

Presentation



```
setEOPannerC:\library\shared\toolbar\graphics\bar
setEOPannerC:\algo.00199.gif i:\api\go
addEOPMenuC:\menu1 HOOK Home default.asp
addEOPMenuC:\menu1 HOOK Library Library/
addEOPMenuC:\menu1 HOOK Downloads downloads/d
addEOPMenuC:\menu1 Code Center code/defa
addEOPMenuC:\menu4 Site Map sitemap/defa
addEOPMenuC:\menu5 HOOK worldwide our/ide

[!--?SCRIPT?--]<HEAD><BODY TOPMARGIN=0 LEFTM
<div id=fbcontainter style=height:50px;-->
<SPAN ID=TDdownlevel>
<TABLE WIDTH=100% CELLPADDING=0 CELLSPACING=0 BORDER
<TR>
<TD VALIGN=TOP HEIGHT=60 HOOKFN=1><A SRC=/default
<TD VALIGN=TOP HEIGHT=60 ALIGN=RIGHT><IMG SRC=/libr
<TD BGCOLOR=000000 HEIGHT=20 VALIGN=MIDDLE ALIGN=C
<FONT COLOR=FFFFFF FACE=verdana,arial SIZE=11>
</div>
</script>
```



```

<? Page Language="VB" ASPCompat="true" %>
<script runat=server>
    Dim myComp
    Public Sub Page_Load()
        myComp = new MySTAComponent()
        myComp.Name = "Job"
    End Sub
</script>
<html>
<|
    Response.Write myComp.SayHello()
%>
</html>

```

```
<script language="C#" runat="server">
protected void Page_Load(Object Src, EventArgs E)
{
    SqlConnection myConnection = new SqlConnection
    ("server=localhost;" +
    "database=pubs;Trusted_Connection=Yes");
    SqlDataAdapter myCommand = new SqlDataAdapter
    ("SELECT * FROM Authors", myConnection);
    DataSet ds = new DataSet();
    myCommand.Fill(ds);
    DataView source = new DataView(ds.Tables[0]);
    MyDataGrid.DataSource = source;
    MyDataGrid.DataBind();
}
</script>
```

The Haskell Days

Swartz, SETL 1970; Burstall and Darlington, NPL 1977; Turner, KRC 1981; Haskell, Python, Scala, ...

Write programs using mathematical ZF set comprehensions syntax.

Wadler 1989

List comprehensions are related to the relational calculus.

Wadler 1992

List comprehensions are a special case of monad comprehensions.

List Comprehensions

Example

```
factors n =  
  [ x | x <- [1..n], n `mod` x == 0 ]
```

select
From
where

```
isPrime n = factors n == [1,n]
```

```
primes n =  
  [ p | p <- [2..n], isPrime p ]
```

The Haskell Days

Swartz, SETL 1970; Burstall and Darlington, NPL 1977; Turner, KRC 1981; Haskell, Python, Scala, ...

Write programs using mathematical ZF set comprehensions syntax.

Wadler 1989

List comprehensions are related to the relational calculus.

Wadler 1992

List comprehensions are a special case of monad comprehensions.

List Comprehensions

Example

```
factors n =  
  [ x | x <- [1..n], n `mod` x == 0 ]
```

select
From
where

```
isPrime n = factors n == [1,n]
```

```
primes n =  
  [ p | p <- [2..n], isPrime p ]
```


List Comprehensions

Simplified translation

```
[ e | True ] =  
  [ e ]  
[ e | q ] =  
  [ e | q, True ]  
[ e | b, q ] =  
  filter b [ e | q ]  
[ e | x <- l, q ] =  
  concatMap (\x -> [ e | q ]) l  
[ e | let decls, q ] =  
  let decls in [ e | q ]
```

Syntactic sugar
over standard list
operations

Monad Comprehensions

Example

```
words :: IO [String]
words =
  do{ putstr "enter a value ..."
      ; x <- getLine
      ; return (words x)
    }
```

Parametrized over
type constructor

```
class Monad m where
  { (>>=) :: m a -> (a -> m b) -> m b
    ; return :: a -> m a
    }
```

Syntactic sugar
over standard
monad operations

List Comprehensions

Simplified translation

```
[ e | True ] =  
  [ e ]  
[ e | q ] =  
  [ e | q, True ]  
[ e | b, q ] =  
  filter b [ e | q ]  
[ e | x <- l, q ] =  
  concatMap (\x -> [ e | q ]) l  
[ e | let decls, q ] =  
  let decls in [ e | q ]
```

Syntactic sugar
over standard list
operations

Monad Comprehensions

Example

```
words :: IO [String]
words =
  do{ putstr "enter a value ..."
      ; x <- getLine
      ; return (words x)
    }
```

Parametrized over
type constructor

```
class Monad m where
  { (>>=) :: m a -> (a -> m b) -> m b
    ; return :: a -> m a
  }
```

Syntactic sugar
over standard
monad operations

HaskellDb Query Monad

```
SELECT X.FirstName, X.LastName  
FROM Authors AS X  
WHERE X.City = 'Oakland'
```

Query
monad

```
oaklands =  
  do{ x <- table authors  
      ; restrict (x!city .==. constant "Oakland")  
      ; project ( au_fname = x!au_fname  
                  , au_lname = x!au_lname  
                  )  
      }
```

intentional representation
for expressions

Haskell Server Pages

XHTML Literals

```
table :: TABLE
table = <TABLE border="1">
    <% mkRows cells %>
</TABLE>
```

Translated to
universal DOM
representation

```
cells :: [[(Int,Int)]]
cells = [ [ (x,y) | x <- [1..16] ] | y <- [1..16] ]
```

```
mkRows :: [[(Int,Int)]] -> [TR]
mkRows =
    map $ \cs -> <TR><% mkColumns cs %></TR>
```

```
mkColumns :: [(Int,Int)] -> [TD]
mkColumns =
    map $ \c -> <TD bgcolor=(color c)><% c %></TD>
```

ASP-style
embedding

Growing C ω



Haskell Server Pages

XHTML Literals

```
table :: TABLE
table = <TABLE border="1">
    <% mkRows cells %>
</TABLE>
```

Translated to
universal DOM
representation

```
cells :: [[(Int,Int)]]
cells = [[ (x,y) | x <- [1..16] ] | y <- [1..16] ]
```

```
mkRows :: [[(Int,Int)]] -> [TR]
mkRows =
    map $ \cs -> <TR><% mkColumns cs %></TR>
```

```
mkColumns :: [(Int,Int)] -> [TD]
mkColumns =
    map $ \c -> <TD bgcolor=(color c)><% c %></TD>
```

ASP-style
embedding

Growing Cω

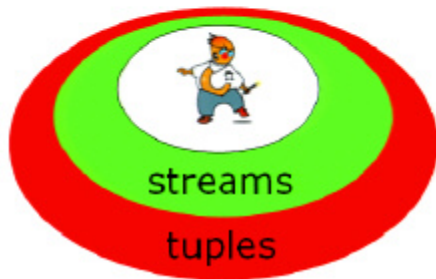


Streams

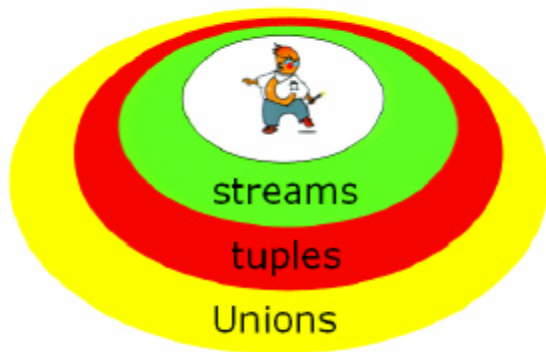
Growing Cω



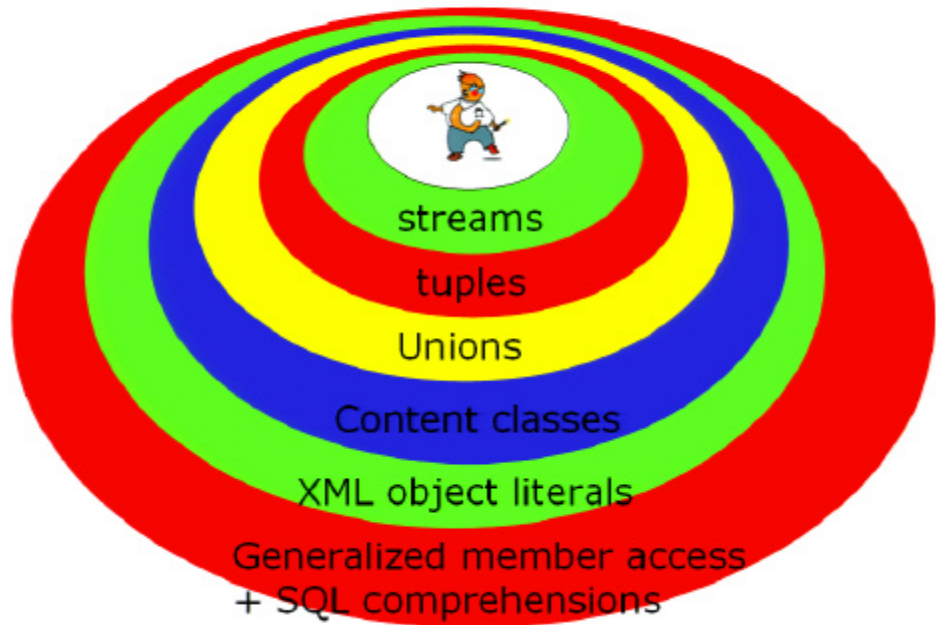
Growing $C\omega$



Growing C ω



Growing Cω



Type System Extensions

(structural)

$T ::= N$
| $T[]$ | $T\{\}$
| $T(\dots, T, \dots)$
| $T|T$ | $T\&T$
| $T!$ | $T?$ | $T+$ | T^*
| $\text{struct } \{\dots, T\ m, \dots\}$

arrays

closures

intersection,
union

streams

tuples (rows)

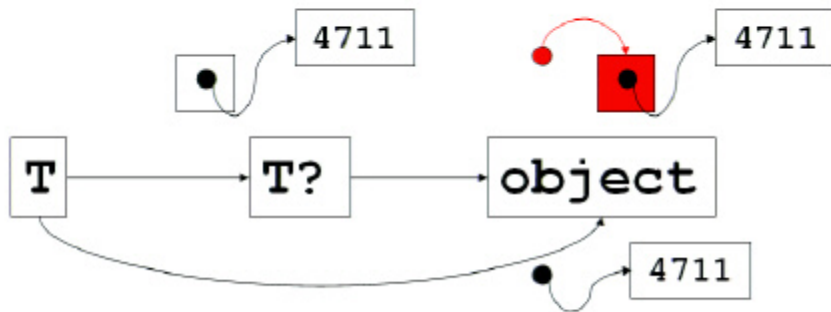
Query Comprehensions

```
string n = "Wolfram";  
struct{String? Subject}* subjects =  
    select it.Subject from it in inbox  
    where it.From == n  
  
foreach(r in  
    select CustomerID, ContactName  
    from dbo.Customers where City == mycity  
    order by ContactName) {  
    ...  
}
```

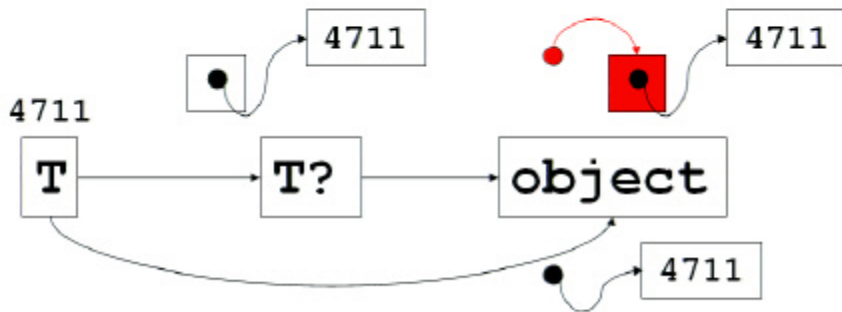
Compiler
plugins

Type inference

Incoherence?



Incoherence?



When up-casting from T? to object
we need to unwrap (→ in Whidbey)

Type System Extensions

(structural)

$T ::= N$
| $T[]$ | $T\{\}$
| $T(\dots, T, \dots)$
| $T|T$ | $T\&T$
| $T!$ | $T?$ | $T+$ | T^*
| $\text{struct } \{\dots, T\ m, \dots\}$

arrays

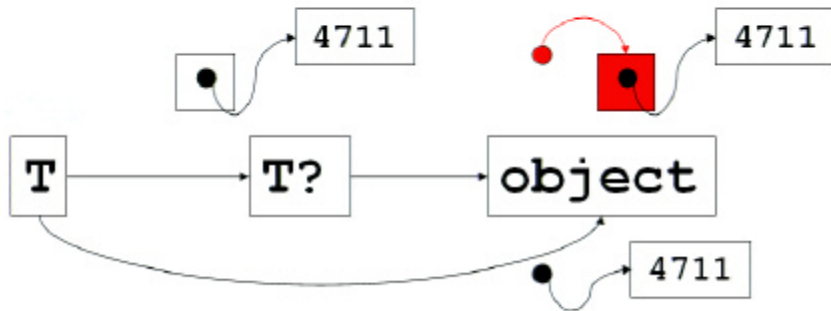
closures

intersection,
union

streams

tuples (rows)

Incoherence?



LINQ Project

VB 9

C# 3.0

...

Standard
Query
Operators

DLinq
(relational)

XLinq
(xml)

LINQ Framework

Query Comprehensions In C# 3.0

```
var contacts =  
    from c in customers  
    where c.State == "WA"  
    select new { c.Name, c.Phone };
```

Type inference

Syntactic sugar
over standard
query operations

```
var contacts =  
    customers  
    .Where(c => c.State == "WA")  
    .Select(c => new { c.Name, c.Phone });
```

Lambda expression

Extension Methods

```
var contacts = IEnumerable<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new { c.Name, c.Phone });
```

```
static class System.Query  
{  
    public static IEnumerable<T>  
        where<T>(this IEnumerable<T> src,  
                Func<T, bool>> p);  
}
```

Extension method for IEnumerable<T>

Expression Trees

```
var contacts = Table<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new { c.Name, c.Phone });
```

```
class Table<T>: IEnumerable<T>  
{  
    public Table<T>  
        where(Expression  
            <Func<T, bool>> p);  
    ...  
}
```

Intensional
representation of
delegate

Extension Methods

```
var contacts = IEnumerable<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new { c.Name, c.Phone });
```

```
static class System.Query  
{  
    public static IEnumerable<T>  
        where<T>(this IEnumerable<T> src,  
                Func<T, bool>> p);  
}
```

Extension method for IEnumerable<T>

Expression Trees

```
var contacts = Table<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new{c.Name, c.Phone});
```

```
class Table<T>: IEnumerable<T>  
{  
    public Table<T>  
        where(Expression  
            <Func<T, bool>> p);  
    ...  
}
```

Intensional
representation of
delegate

Query Comprehensions

```
string n = "Wolfram";  
struct{String? Subject}* subjects =  
    select it.Subject from it in inbox  
    where it.From == n  
  
foreach(r in  
    select CustomerID, ContactName  
    from dbo.Customers where City == mycity  
    order by ContactName) {  
    ...  
}
```

Compiler
plugins

Type inference

HaskellDb Query Monad

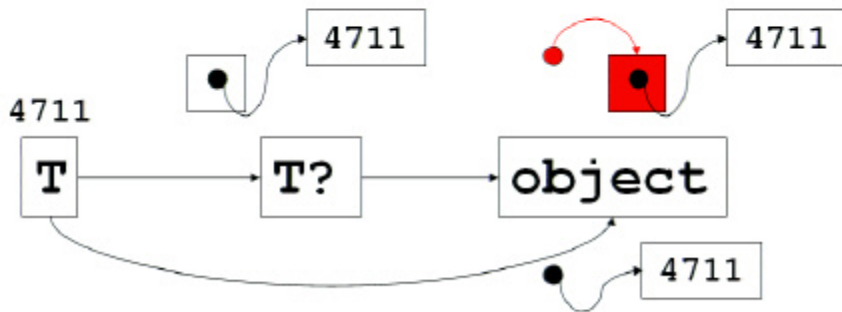
```
SELECT X.FirstName, X.LastName  
FROM Authors AS X  
WHERE X.City = 'Oakland'
```

Query
monad

```
oaklands =  
  do{ x <- table authors  
      ; restrict (x!city .==. constant "Oakland")  
      ; project ( au_fname = x!au_fname  
                  , au_lname = x!au_lname  
                  )  
      }
```

intentional representation
for expressions

Incoherence?



Expression Trees

```
var contacts = Table<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new { c.Name, c.Phone });
```

```
class Table<T>: IEnumerable<T>  
{  
    public Table<T>  
        where(Expression  
            <Func<T, bool>> p);  
    ...  
}
```

Intensional
representation of
delegate

Anonymous types, Object initializers

```
var contacts =  
    from c in customers  
    where c.State == "WA"  
    select new { c.Name, c.Phone };
```

Anonymous
types

Object Initializers

```
var Joe = new Person{  
    Name = "Joe", Age = 42,  
    Address = { Street = "1th st",  
                City = "seattle" }  
}
```

Constructor inference

Expression Trees

```
var contacts = Table<Customer>  
    customers  
    .where(c => c.State == "WA")  
    .select(c => new { c.Name, c.Phone });
```

```
class Table<T>: IEnumerable<T>  
{  
    public Table<T>  
        where(Expression  
            <Func<T, bool>> p);  
    ...  
}
```

Intensional
representation of
delegate

Anonymous types, Object initializers

```
var contacts =  
    from c in customers  
    where c.State == "WA"  
    select new { c.Name, c.Phone };
```

Anonymous
types

Object Initializers

```
var Joe = new Person{  
    Name = "Joe", Age = 42,  
    Address = { Street = "1th st",  
                City = "seattle" }  
}
```

Constructor inference

Query Comprehensions In Visual Basic 9

SQL-style Select-From-Where
vs Yoda-style from-where-select

```
Dim contacts =  
    select c.Name, c.Phone  
    From c In customers  
    where c.State = "WA"
```

Parity wrt to other C# 3.0
features

```
from c in db.Customers
where c.City == "London"
select
    new { c.Name, c.Phone }
```

Application

LINQ Query

DLinq

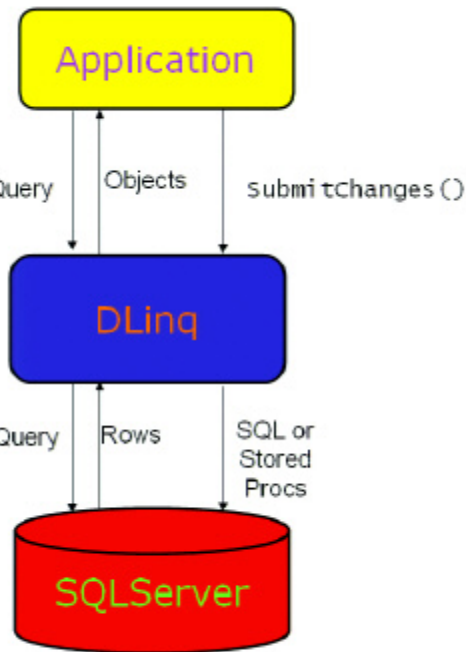
SQL Query

```
select Name, Phone
from customers
where city = 'London'
```

SQLServer



```
from c in db.Customers
where c.City == "London"
select
new { c.Name, c.Phone }
```



```
select Name, Phone
from customers
where city = 'London'
```

```
from c in db.Customers
where c.City == "London"
select
    new { c.Name, c.Phone }
```

Application

LINQ Query

Objects

submitChanges()

DLinQ

Services:

- Change tracking
- Concurrency control
- Object identity

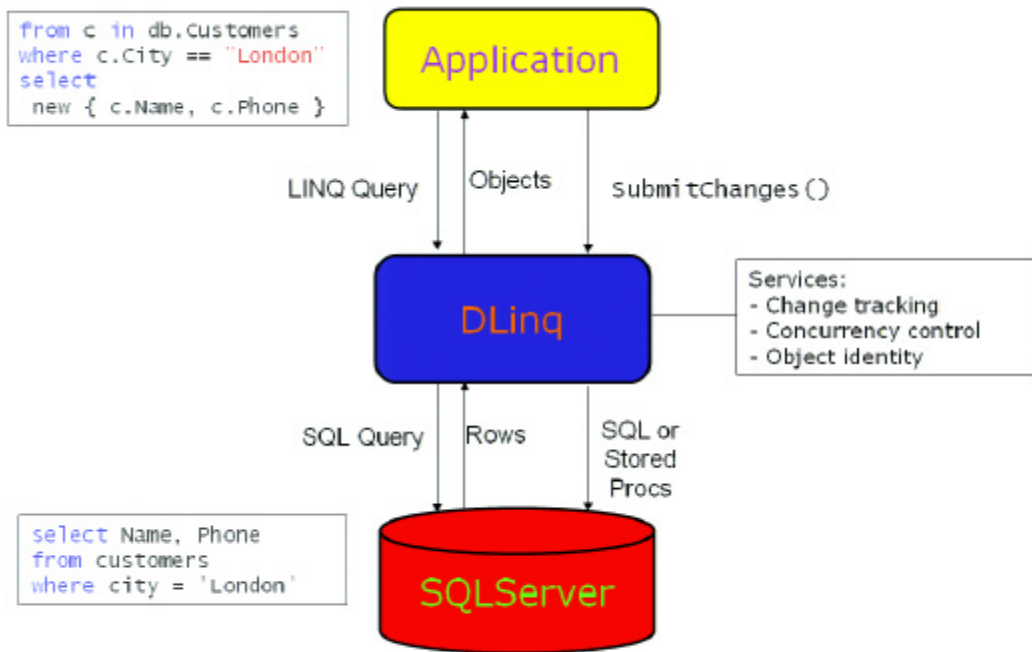
SQL Query

Rows

SQL or
Stored
Procs

```
select Name, Phone
from customers
where city = 'London'
```

SQLServer



DLinQ

O/R Mapping

Mapping through attributes

Manually authored or tool generated
DataContext allows access to relational data as
objects

Automatic change tracking

Auto-generated updates using optimistic
concurrency

Works with existing infrastructure (ADO.Net)

Integrates with System.Transactions, SQL pass-
through, returning objects from SQL queries.

XML DOM

```
Dim PO As New XmlDocument

Dim purchaseOrder As XmlElement = _
    PO.CreateElement("purchaseOrder")
PO.AppendChild(purchaseOrder)

Dim orderDate As XmlAttribute =
    PO.CreateAttribute("orderDate")
orderDate.Value = "1999-10-20"
purchaseOrder.Attributes.Append(orderDate)

Dim shipTo As XmlElement =
    PO.CreateElement("shipTo")
purchaseOrder.AppendChild(shipTo)

Dim country As XmlAttribute =
    PO.CreateAttribute("country")
country.Value = "US"
shipTo.Attributes.Append(country)
```

What
does
this
program
do?

XLinq API

Functional (expression-based) construction

```
Dim Item = _  
New XElement("item", _  
    New XAttribute("partNum", "926-AA"), _  
    New XElement("productName", "..."), _  
        New XElement("quantity", 1), _  
        New XElement("price", 39.98), _  
        New XElement("shipDate", "1999-05-21"))))
```

Simple

Context-free (no document scope)

XML DOM

```
Dim PO As New XmlDocument

Dim purchaseOrder As XmlElement = _
    PO.CreateElement("purchaseOrder")
PO.AppendChild(purchaseOrder)

Dim orderDate As XmlAttribute =
    PO.CreateAttribute("orderDate")
orderDate.Value = "1999-10-20"
purchaseOrder.Attributes.Append(orderDate)

Dim shipTo As XmlElement =
    PO.CreateElement("shipTo")
purchaseOrder.AppendChild(shipTo)

Dim country As XmlAttribute =
    PO.CreateAttribute("country")
country.Value = "US"
shipTo.Attributes.Append(country)
```

What
does
this
program
do?

XLinq API

Functional (expression-based) construction

```
Dim Item = _  
New XElement("item", _  
    New XAttribute("partNum", "926-AA"), _  
    New XElement("productName", "..."), _  
        New XElement("quantity", 1), _  
        New XElement("price", 39.98), _  
        New XElement("shipDate", "1999-05-21"))))
```

Simple

Context-free (no document scope)

```

Dim PO = <purchaseorder orderDate=(System.DateTime.Today)>
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <%= BillTo %>
    <items>
        <%=
            select <item partNum=(O.PartID)>
                <productName>
                    <%= O.Product %>
                </productName>
                <quantity>
                    <%= O.Quantity %>
                </quantity>
                <price>
                    <%= O.Price %>
                </price>
            </item>

            From O In Orders
            where O.Name = "Robert Smith"

        %>
    </items>
</purchaseorder>

```

Translated to
XLINQ constructor
calls

ASP-style
embedding

XLinq API

Functional (expression-based) construction

```
Dim Item = _  
New XElement("item", _  
    New XAttribute("partNum", "926-AA"), _  
    New XElement("productName", "..."), _  
        New XElement("quantity", 1), _  
        New XElement("price", 39.98), _  
        New XElement("shipDate", "1999-05-21"))))
```

Simple

Context-free (no document scope)

XML DOM

```
Dim PO As New XmlDocument

Dim purchaseOrder As XmlElement = _
    PO.CreateElement("purchaseOrder")
PO.AppendChild(purchaseOrder)

Dim orderDate As XmlAttribute =
    PO.CreateAttribute("orderDate")
orderDate.Value = "1999-10-20"
purchaseOrder.Attributes.Append(orderDate)

Dim shipTo As XmlElement =
    PO.CreateElement("shipTo")
purchaseOrder.AppendChild(shipTo)

Dim country As XmlAttribute =
    PO.CreateAttribute("country")
country.Value = "US"
shipTo.Attributes.Append(country)
```

What
does
this
program
do?

```

Dim PO = <purchaseorder orderDate=(System.DateTime.Today)>
    <shipTo country="US">
        <name>Alice Smith</name>
        <street>123 Maple Street</street>
        <city>Mill Valley</city>
        <state>CA</state>
        <zip>90952</zip>
    </shipTo>
    <%= BillTo %>
    <items>
        <%=
            select <item partNum=(O.PartID)>
                <productName>
                    <%= O.Product %>
                </productName>
                <quantity>
                    <%= O.Quantity %>
                </quantity>
                <price>
                    <%= O.Price %>
                </price>
            </item>

            From O In Orders
            where O.Name = "Robert Smith"

        %>
    </items>
</purchaseorder>

```

Translated to
Xinq constructor
calls

ASP-style
embedding

Late binding over XML

Child axis

BillTo.street

→ BillTo.Elements("street")

Attribute axis

BillTo.@country

→ BillTo.Attribut("country")

Descendants axis

PO...item

→ PO.Descendants("item")

Late binding over XML

Child axis

BillTo.street

→ BillTo.Elements("street")

Attribute axis

BillTo.@country

→ BillTo.Attribute("country")

Descendants axis

PO...item

→ PO.Descendants("item")

Static Duck Typing (typeclasses--)

```
Extension Customer For XElement
  Property Street As String
  Get
    Return CStr(Me.Element("street"))
  End Get
End Property
  Property country As String
  Get
    Return CStr(Me.Attribute("street"))
  End Get
End Property
  Property Street As IEnumerable(of Item)
  Get
    Return CType(Me.Descendants("item"), Item)
  End Get
End Property
End Extension
```

Named extension

Compile-time cast

A decorative grid pattern with thick black lines. The grid is composed of several squares. The top-left square is red. The bottom-left square is yellow. The bottom-right square is blue. The rest of the grid is white.

Conclusion

Programming Language
Theory is relevant.
But it requires at least 20
years of patience and
ripening.